Middle East Technical University
Department of Computer Engineering

CENG 477
Introduction to Computer Graphics

Fall 2007 - 2008
Project

REGULATIONS

Team: The project has to be done by groups of exactly 2 partners. Each one of the partners may receive different grades from the project phases.

Submission: At each phase, each group will submit only one copy using cow.ceng.metu.edu.tr.

Late Submission: Late submission is allowed **only for one day** with a penalty of 20 points.

Cheating: In case of cheating, all parts involved (source team(s) and receiver team(s)) will get zero.

Updates: The information contained in this document is subject to change.

Newsgroup: You **must** follow the newsgroup **for discussions and possible updates**.

Programming Language: C or C++

Libraries: OpenGL, GLUT, GLUI (optional), Object3DS (A simple 3ds Loader, see Appendix, part a)

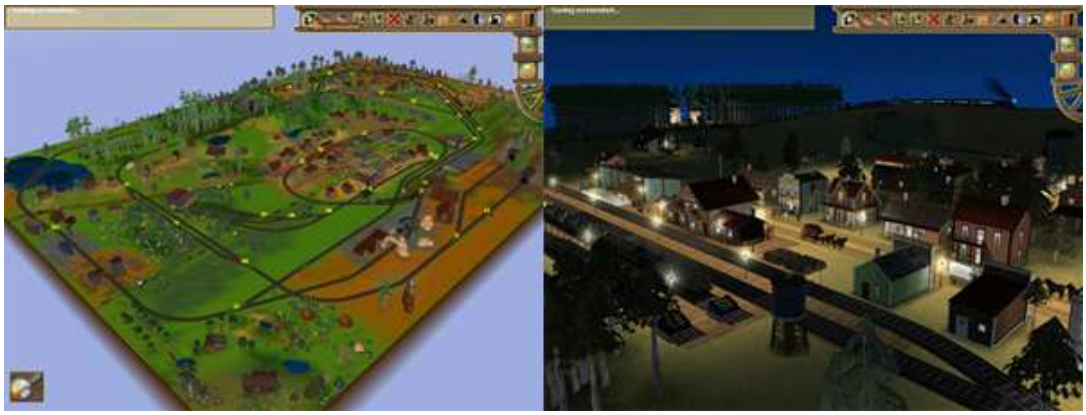
Environment: Linux

GENERAL PROJECT DESCRIPTION

In this project you will implement an editor and a basic animation tool for management of a train. Mainly, in the project, there will be a train on a railroad which is already defined by many rail tracks in a simple environment.

The project is composed of three separate phases in which you will implement respectively in 2D, in 3D and in 3D with advanced techniques. This document explains the specifications of the project in detail and unless otherwise stated, all those specifications are compulsory for each group.

The main idea of the project is to create an adjustable train simulation. There are many advanced examples of this idea in the market of games whose examples are given in the below figures (respectively from Iron Horses, Microsoft Train Simulator, AIT Trains). These advanced examples and most of the figures in this document are not given as a restrictive specification on what you have to do, but just to create an imagination of what could be achieved.



RECITATIONS

In order to give more details and some guidance on the specifications, to clarify the existing specifications and to answer your questions, for each phase there will be a recitation. Participating in these recitations will ease your work on the project. We advise all of you to attend these recitations, but it is a **must** that at least one person from each project team attends the recitation. You will be responsible from everything covered in these recitations.

SUBMISSION DETAILS

For each phase, you will submit a file named phasex.tar.gz (e.g. phase1.tar.gz). This submission package **must** include the following:

1. All of your **source codes**
2. An appropriate **makefile** (which creates an executable named as phasex, e.g. phase1)
3. All necessary files for additional **libraries**; as an example, if you use GLUI for the project, your submission package must include glui.h and libglui.a.
4. One **screenshot** named as phasex.jpg (e.g. phase1.jpg); choose a screenshot which reflects nearly all implemented capabilities of the related phase.
5. Saved scenarios (optional); this may be helpful in demonstrations
6. **Phase report**; a single text file named as readme.txt. A template and a sample phase report will be provided. The format of the readme.txt is as follows (you have to obey this strictly):
   - A single line for names and IDs of project members
   - A single short paragraph presenting some general information about your project, your status at the current phase or anything similar you can think of.
   - A list of current functionalities of your project; each functionality will be presented at a single line with its user interaction techniques:

     <User interaction techniques> : <Short description of the functionality>

We will also provide a simple script which we will use in evaluating your projects. You **have to** try that script with your submission package before submission.

DEMONSTRATIONS

For each phase, you will make a demonstration to your assistants.

- The demonstration will take place in a linux environment (the assistant's computer or in case of a problem one of the computers from department's laboratory), no other choice is possible. We advise you to be sure that you try your project in department laboratories before each submission.
- Each team member must attend the demonstrations.
- For the demonstrations, you may include a saved scenario in your submission package and load it to present some of your capabilities. (optional)
- The demonstration schedule will be arranged after submissions of each phase.

SOME GENERAL ISSUES ABOUT THE PROJECT

For all three phases of the project, you have to consider the following common specifications:

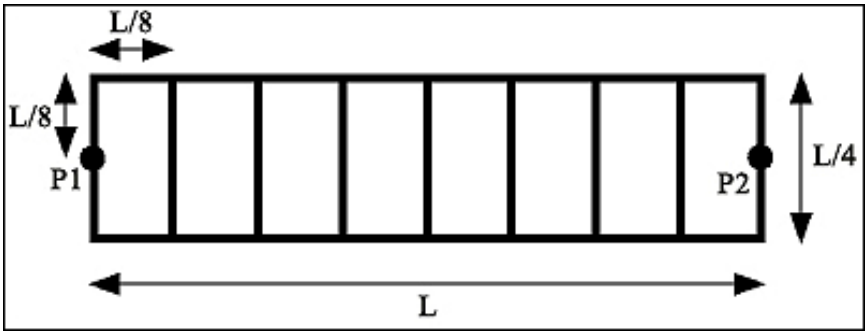
- **Save & Load Functionality:** At each phase, you have to provide the functionality to save and load the current design and sometimes the current states in your project. You will need to store two files; one for the railroad definition (railroad.txt) and one for the other data (any name you wish). We are providing you a mandatory file format to define the railroad. However, you are free on the file format for the other data to be saved and loaded. (**[Hint]**: In general, you have to be careful on your data structure choices and consider all phases with all functionalities.)
- **Preserving Previous Phases:** Beginning with the second phase, you have a requirement to preserve the functionalities and the views from the previous phases. For example, when you finish second phase, designing in 2D and the related 2D view must still be available. You can supply this by using a mode change, using multiple windows or sub-windows.
- **Logging for Debug Purposes:** As you have already experienced in your warmup homework, for graphics applications logging (in a simple definition printing some messages) is a critical issue and event logs can help you to debug your codes more easily. In the project, you are expected to log some events and you are expected to decide the events that you will log.
- **General User Interface and Interaction:** Unless otherwise stated, you are free to choose the type of user interaction and the related user interface elements for each functionality. However don't forget that user interaction is a critical issue and your design at this point is important for the project.
- **GLUI (optional):** You can use GLUI in your project for managing user interface elements and user interaction. However, you **may** have some difficulties due to some unresolved bugs in GLUI. Therefore, you may also prefer not using GLUI and you may cover user interaction requirements by alternative ways (an example is using a keyboard key or a GLUT menu instead of a GLUI button).
- **Some General Graphics Issues:** The following are some examples of what you can easily implement to make your project better by means of displayed graphics and user interaction:
    - Filter (display & hide) certain elements in the environment, this is totally similar to what you have already done as toggling in the warmup homework. (e.g. display & hide railroad tracks in the environment)
    - Use different colors/coloring to make things and the process more understandable. (e.g. different colors for a selected and an unselected element, etc.)
    - Display some labels to aid the user at any step (e.g. when a new element is added to the environment display its coordinates, etc.)

Note: All of the above specifications will also be considered in grading your projects.

PHASE 1: The Rookie Machinist (Due Date: 29.11.2007)

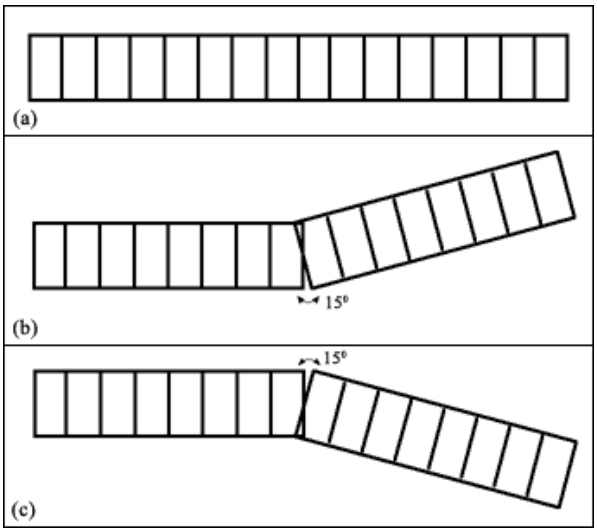
In The Rookie Machinist phase, you will implement the 2D capabilities of the project. From a bird's eye view, you will create an environment, design a railroad, locate a train and move the train on the railroad.
The following specifications must be implemented in The Rookie Machinist Phase of your project:

- The 2D part of your project will be composed of two main modes:
  - The first mode is the editor mode in which you can change the environment, the railroad and the train.
  - The second mode is the basic animation mode in which the train will move by following the railroad. At this mode, the user can change the state of the railroad switches, but editorial capabilities are not available.
- The user can switch between these two modes.
- The editor mode is mainly composed of three main groups of functionalities; environment related, railroad related and train related.
- Environment related functionalities are mainly used to create an environment by using three types of elements; buildings, tree and street lights.
  - Firstly, the user must be able to initiate a railroad by selecting a point P on the ground, and then add new tracks to the railroad. The first track of the railroad will be located on P and placed horizontally (at 0° with the horizontal axis).
  - An environment element (a building, a tree or a street light) can be added by selecting a position on the ground.
  - In 2D, a building is represented by a blue rectangle labeled 'B', a tree is represented by a green triangle labeled 'T' and a street light is represented by a small red rectangle labeled 'L'.
  - An environment element can be selected and deselected.
  - A selected environment element can be deleted or rotated.
  - Street lights will be illuminated in the last phase.
  - For the sake of simplicity, you can use a few buildings to create a station. Such a composition does not have any functionality and you need not elaborate this much.



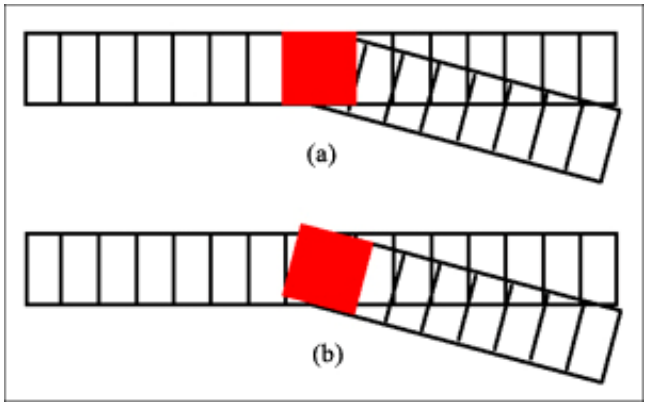
- Railroad related functionalities are used to build up a railroad by using consecutive railroad tracks and railroad switches.

- There can be only one railroad in the environment. Branches are possible on this railroad by using railroad switches, but unconnected components can not be created.
- Each railroad track is identical by means of their dimensions. A track has a length of L and a width of L/4. P1 is the middle point at the left end of the track and P2 is the middle point at the right end of the track. 2D representation of a track is shown in the above figure (P1, P2 and dimensions are given for illustrative purposes)
- The track length, L, will be used as a general parameter to define various proportions for this project. You can choose any value of L, but if you want to deal with integers always, you need $L = 2^n$ for $n>4$.
- A track can be selected and deselected.
- A last track is a track on the railroad which doesn't have any other tracks after it; in other words no tracks have been added to its right end. There can be more than one last track on a railroad, as there are switches on the railroad.
- The user will be able to add a new track to the right end of a selected last track in the railroad. By means of user interaction, the process will be as follows:
    1. A last track will be selected by a mouse click
    2. The direction of the new track will be selected by the user by a method that you will find.
    3. The new track with the chosen direction will be added to the right end of the selected last track, at point P2. The selected last track and the new track are now connected.
- There are three possible directions for a new track;
    1. It can be at the same direction with the previous track (a)
    2. It can turn left (counterclockwise) by 15^o around an axis perpendicular to the ground (b)
    3. It can turn right (clockwise) by -15^o around an axis perpendicular to the ground (c)



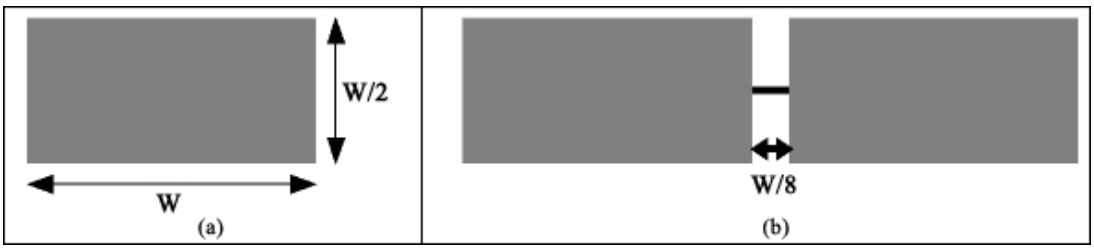
- If a track is a last track and it is selected, then it can be deleted. Only last tracks can be deleted.
- You should better represent each track by two points; P1 and P2 (mid points of short edges at each end). You may need to store positions (x and y coordinates) of these two points for each track in your railroad.
- A railroad switch connects one track to exactly two tracks. The switch is represented as a square whose edges have L/4 length.
- The user can add a railroad switch to a right end of a track. When a switch is added then the related track can be connected to two tracks. At this point, the user has to add these two tracks before any other construction on the railroad.
- A switch is deleted if its preceding track is deleted. (To delete the preceding track, the user must have already deleted both of the ensuing tracks)

- Orientation of a switch is dependent on the ensuing tracks and its current state; it is oriented in the direction of the current track to be followed.
- User must be able to change the state of a switch by a middle mouse click on the switch. A switch has exactly two states; each state means that the train will follow one of the ensuing tracks. A sample is shown in the figure on the right.



- The railroad can have closed curves, but it can also have dead ends (which are actually last tracks) which have no connection to other tracks. Branches can also be connected to the railroad in an appropriate position and make up a closed curve.
- In order to form closed curves, the user can define connections. A connection can only be defined if P2 of a track intersects with P1 of the other track and angle between these two tracks is 15, 0 or -15. These are named as user-defined connections. Remember that consecutive tracks are connected automatically.
- The railroad must be stored in a single file which extends the idea of L-systems and turtle graphics. Logic behind this idea, format and details of this file is given in Appendix, part b.
- For all railroad definition, you can assume the user will always input appropriate data. For example; the tracks will never intersect, switches will not be very close to each other, etc.
- Railroad always has only one direction for movement (which starts from the initial position and direction), switches and connections do not violate this.

- Train related functionalities are used to define a train on the railroad.
  - There will be always one train on the railroad.
  - A train is composed of one locomotive and many wagons, in general which are called railroad cars.



  - In 2D representations, each of the wagons and the locomotive are identical by means of shape and dimensions. They are a box of length W and width W/2. The center of this box is used as the reference point for all kinds of transformations (e.g. translation, rotation) of this box.
  - W is dependent to L; W=L/2
  - For two consecutive railroad cars, the distance between them is W/8 in normal case; when both have the same orientation. In other cases, this value will be used as an assumption of the distance on the tracks.
  - A locomotive will have label of ‘L’ on its box.

- There are two kinds of wagons, passenger wagons and goods wagons. Passenger wagons have a label of ‘PW’ and goods wagons have ‘GW’ on their boxes.
- Each railroad car can have a different color. For easiness, you should better keep a color state for railroad cars.
- As stated a wagon can be passenger wagon and goods wagon; you should better keep a state for determining the type of the wagon.
- Current color state and wagon type state need initial values and a simple method for the user to change the states.
- Current color state and wagon type state will be used whenever a new railroad car is added to the train.
- The user will be inserting the train; locomotive and the wagons, on the railroad starting from the locomotive.
- The user will select a track on the railroad to place the locomotive. The center of the locomotive will be located in the middle of the selected track and in the same orientation with this selected track.
- After locating the locomotive, the user will not specify locations for any other wagons. Instead, new wagons will be inserted just after the last railroad car as the user requests to add a wagon. Once more, take into account the orientation of the railroad track while you are placing the new wagons and be sure they stay in the railroad. (You can assume that the user will not try to add a new wagon, if the railroad ends after the last wagon)
- The user can delete the last wagon in the train.
- The user will be able to select and deselect the railroad cars.

- You must provide a simple grid, which you will display on the ground, to help the user understand the positioning of the objects. The user must be able to select to view or hide the grid.
- In the basic animation mode, you will implement the movement of the train by following the railroad.
  - Once the path and the train are inserted, the user may switch to the basic animation mode.
  - The user has three main abilities on the animation at this mode:
    - Start the animation; the train will start moving on the railroad
    - Stop the animation; the train will stop and stay at its place
    - Initiate the animation; the train will be relocated to its initial position (Initial position is the place of the train from the editor mode)
  - During the animation, the user must be able to change the train's route on the railroad by changing the states of the switches.
  - If the railroad ends at some point, the train will stop there.
  - The train has to follow the next track for the user-defined connections.
  - You will need an algorithm for the train to follow the railroad. You are free on your decision, except some constraints:
    - You must divide each railroad track to 32 pieces for animation. Position of the s^{th} piece can be found as follows: (P1 and P2 are defined above in tracks, P1_x represent x coordinate of P)

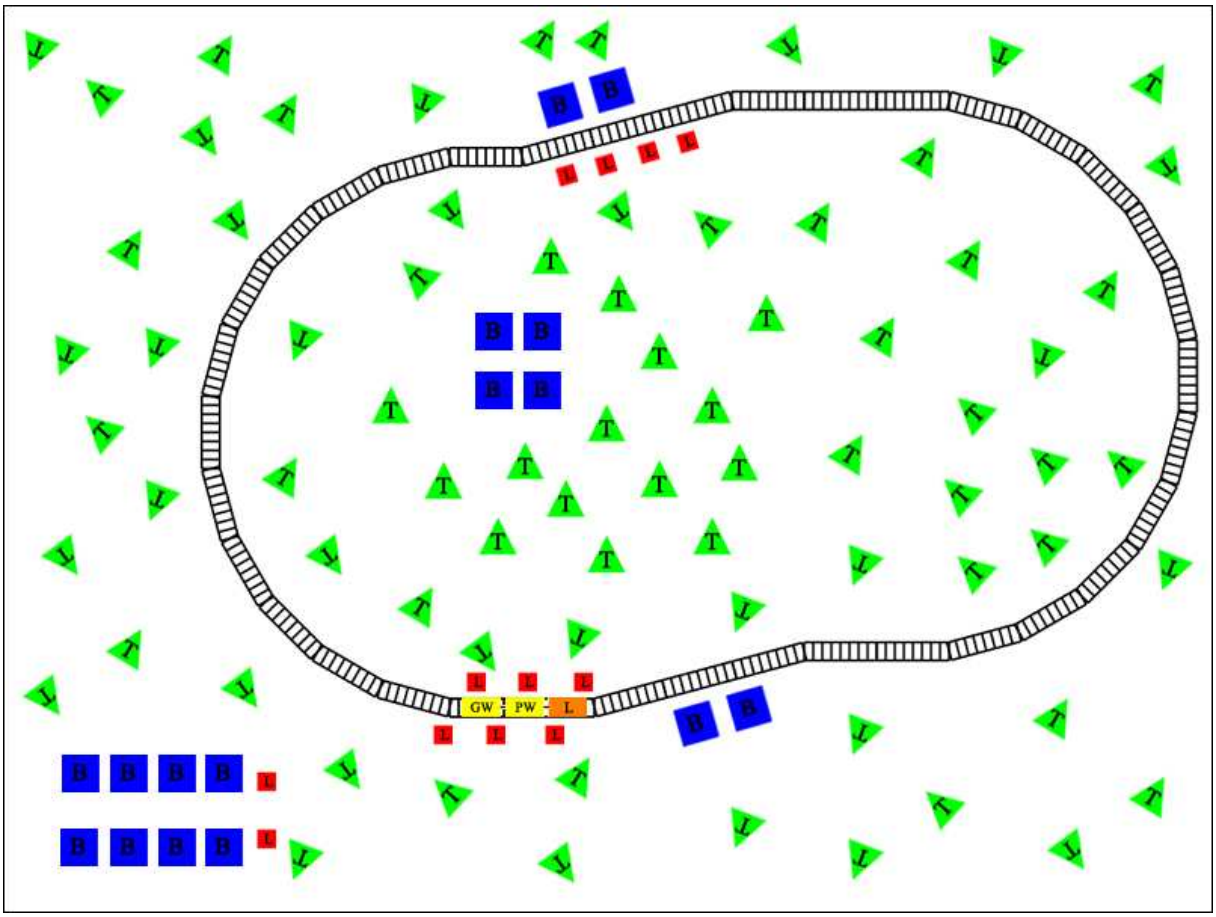
Position(s) = (((P2_x – P1_x)/32)*s, ((P2_y – P1_y)/32)*s)

- You have to use a small frames per second value, not greater than 16.

- You will be provided with the details of a sample algorithm in the recitation. This sample algorithm is easy to implement, so we remind you attend the recitation again.

- We advise you to have a large window for this phase, not smaller than 800*600 and not larger than 1024*768. Anyway, such area may not be enough for your environment and railroad. So, you have to implement some kind of sliding mechanism (slide the area that the user can see in your environment by adjusting the coordinate system). Choose an integer k and perform the following with the given keyboard commands:
  - ‘a’ : Move left by horizontally k coordinates
  - ‘d’ : Move right by horizontally k coordinates
  - ‘w’: Move up by vertically k coordinates
  - ‘s’ : Move down by vertically k coordinates

- The below figure is an illustration of a possible snapshot for this phase. This is not a direct example of what your program should do; however this figure is given in order to give you an idea of this phase.

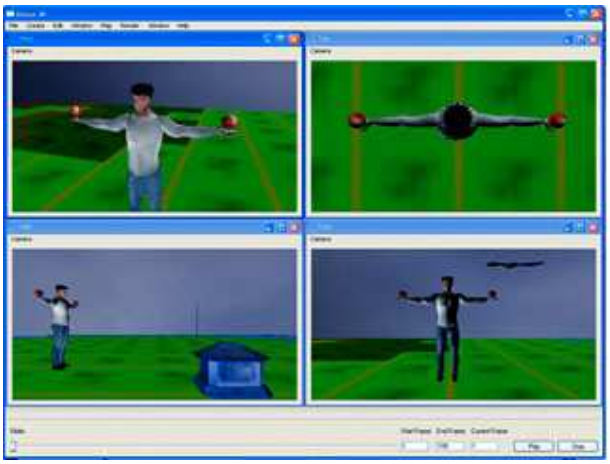


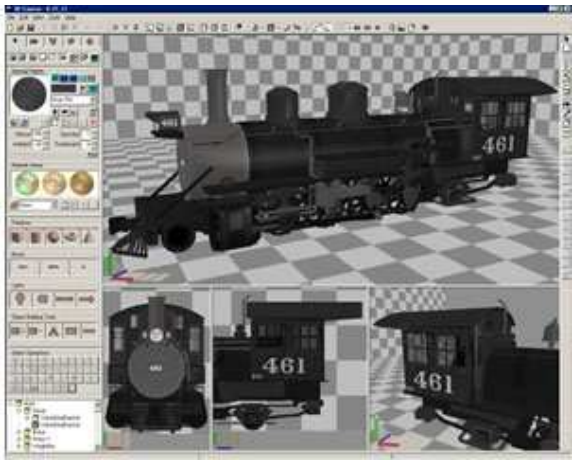
Railroad definition: “tt+ttt-tt+t+t+t+t+t+t+t+t+t+t+t+t+ttt+ttt-t+t+t+t+t+t+t+t+t+t+t+t+te”

PHASE 2: The Veteran Machinist (Due Date: 18.12.2007)

In The Veteran Machinist Phase, you will enlarge the world into the 3rd dimension. For this purpose, you will be implementing some 3D capabilities of the project.
The following specifications must be implemented in The Veteran Machinist Phase of your project:

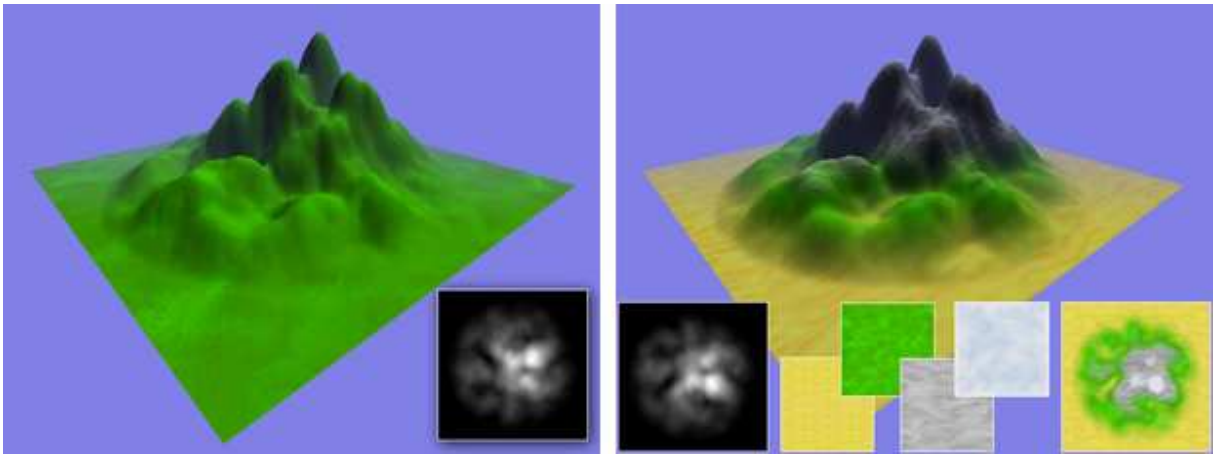
- As mentioned, the main goal in this phase is to render the world in 3D. For this purpose, for everything in your environment you have to provide a logical representation in 3D. Mainly, a logical representation for an object in 3D can be achieved by satisfying that the object has a decent height, the object has some components for being realistic and the object has been situated in a realistic and reasonable manner.
    - Samples for logical 3D representations for the main objects of this project topic are provided in this document at the end of Phase 2 specifications. These images are just samples of what you may provide, so you are free to create your own designs for these objects. However, in such designs you have to include each of the listed compulsory components. You are free to decide on their number and shape for each object.
    - Samples are not given for railroad related objects (i.e. railroad tracks, switches). For those, there are no compulsory components and you are free in your designs. The simplest design that is acceptable is just to give height to these objects (in other words use rectangular prisms instead of rectangles).
    - For each object you should better create the 3D definition and render it by using OpenGL Display Lists.
- Starting from this phase, you will provide the user three main views; 2D view of Phase 1, perspective view and mobile camera view. The user may either switch between these three views or view them at the same time by using multiple windows or sub-windows. Examples of multiple windows and sub-windows are given in the below figure.





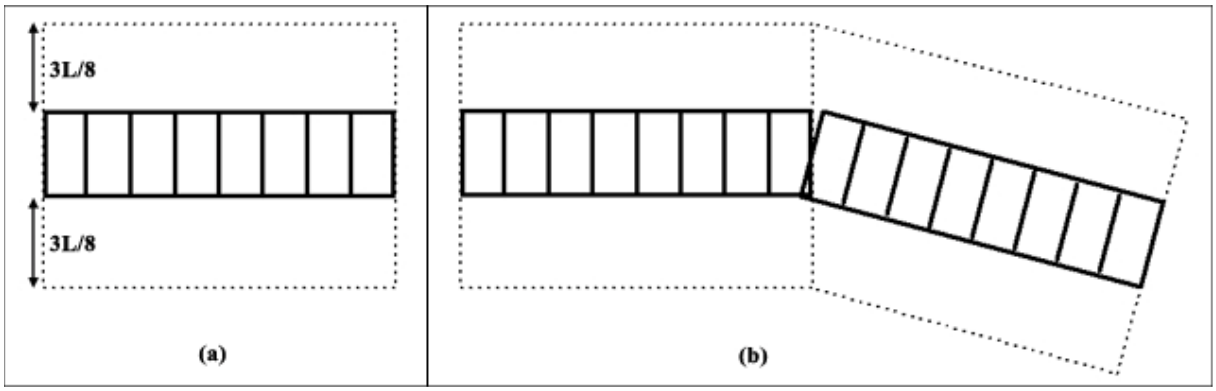
- Changes that have been done in 2D view of Phase 1 must be reflected to other views at the same time.
- The first view is 2D view of Phase 1. As stated in the general issues title, you have to preserve all the capabilities of Phase 1 in this phase. All of these capabilities will be provided to the user in this view.

- The second view is the perspective view. Many tools that are being used widely nowadays provide different views from different perspectives as a common facility. In this project, you will provide one perspective view from a high position which will enable the user to see nearly all the environment at one glance.
- The last view is mobile camera. For this purpose, you will implement basic camera capabilities, so that we can wander in the environment. You are expected to implement all of the following camera options:
    - The camera has a fixed position with respect to the train. In other words, the camera moves as the train moves. The critical point is that the camera will also follow the railroad like a virtual railroad car. Here, you must implement three main places that it can be located at:
        i. At the front of the train, looking forwards and positioned at the front of locomotive
        ii. At the front of the train, looking backwards to the train and positioned W distance in front of locomotive
        iii. On one side (left or right) of the train, looking sideways to the fields and positioned on a selected wagon (The user must have selected a wagon in 2D view of Phase 1 before operating this camera mode)
    - The camera is independent of the train and it simulates a walk-through on the terrain. Think of this as if it is the eyes of a person, which the user can move around the environment using an interface that you will define. The user can move forward, move backward, turn left and turn right. For this purpose you have to keep the camera at a constant height with respect to the terrain. You should also be able to rotate its view upwards-downwards.
- In this phase, the environment that you use will be a 3D terrain instead of a planar surface. You will create the terrain by using a height map. In context of terrain rendering, a height map is a raster image which stores elevation values at each pixel. Simply, each value represents the elevation of a point in a 2D grid. By using elevation data, the position of each point is found and these points are triangulated to create the terrain.

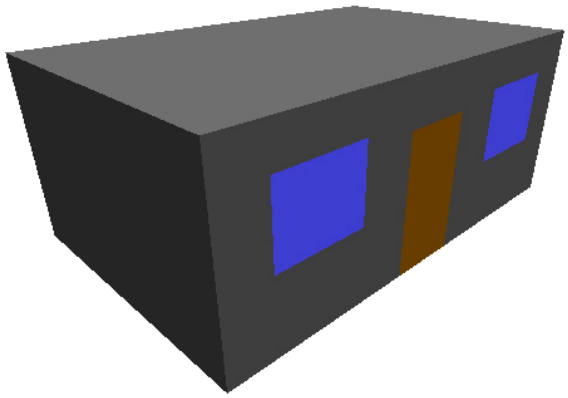
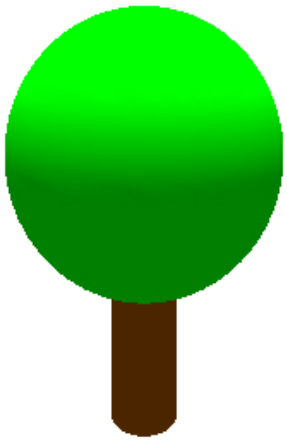
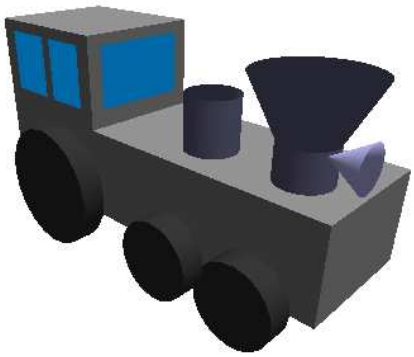
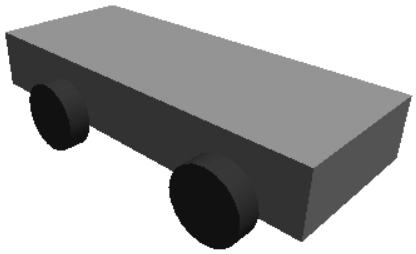
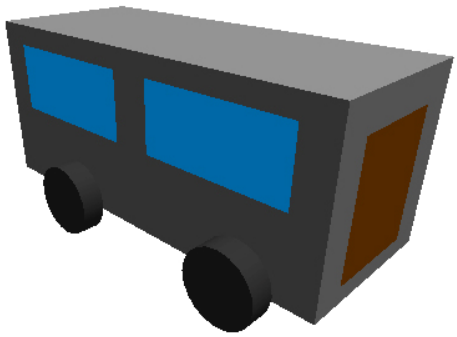


- When you create a terrain, you may need to adjust your environment and objects in order to situate your objects realistically on the terrain. The adjustment is sometimes just changing the elevation value for the related object. However sometimes this adjustment is a more complex operation like manipulating the environment. In this project, we classify environment objects into two categories about this issue:
    - Objects need simple adjustment: Buildings, Trees, Street Lights
    - Objects need complex adjustment: All train and railroad related objects

- For placing all train and railroad related objects on the terrain; your approach will be somehow excavating the terrain instead of adjusting the positions of the objects. It is known that all train and railroad related objects are on a path defined by the railroad. So as a solution, it will be enough if you just manipulate parts of the terrain that intersect with the railroad. The manipulation is simple; you will just change the related elevation data to ground value (i.e. assign 0). For better results, you need a larger intersection area than just the railroad's width. For this purpose, we defined the intersection area as given in the below figure.



- You can calculate coordinates of intersection areas (in other words larger quads) by using P1 and P2 definitions of railroad tracks. For each quad, two vertices are calculated from P2 of previous track and the other two vertices are calculated by P2 of current track.
- You will be reading height maps from .ppm files. Each file will include 65*65 elevation values that range between 0 and 255. The user must have the ability to load and render any height map defined in this format.
- As 65*65 is not a high resolution, you will be scaling the height map to your environment. In other words, one grid cell in the height map will be covering a large area in your environment. For this purpose, you need a scale value, namely k. You are free to choose the value of k, but we advice $k \geq L$.
- In order to render the terrain, you need to create triangles from the height map data. Each grid cell in the height map should be represented by two triangles. As we have a high k value, our triangles will create a terrain which will be a bit rough by means of smoothness.
- The intersection areas defined above will be overlapping with some grid cells in the height map data. In order to find these grid cells, you will be using an algorithm which is effective and easy to implement. However, results of this algorithm will be rough like our terrain.
- For each intersection area, you will find border lines in the grid definition of the height map. Mathematically, you will find maximum and minimum grid lines which are the most closest to the intersection area in both vertical and horizontal dimensions of the grid. These will be the boundaries and you will just change all elevation values inside these boundaries to a ground value (0 for our format definition).

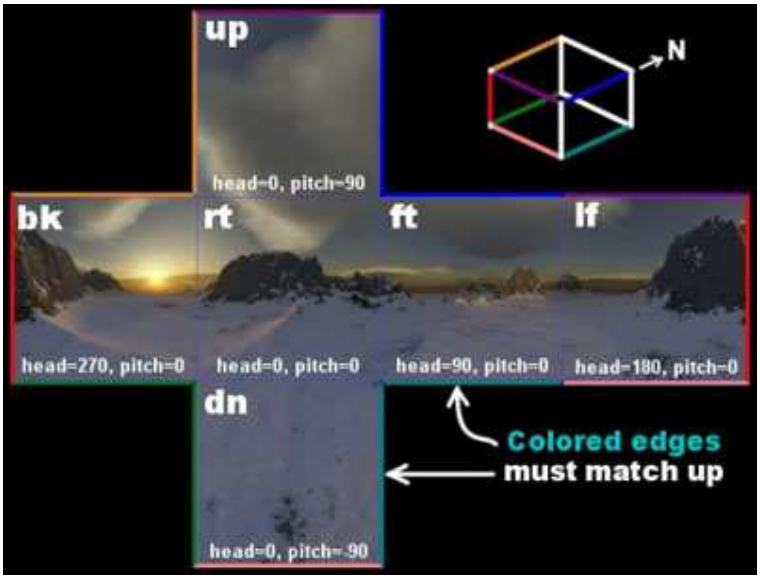
<table>
<tr>
<td> 
Building
Compulsory Components: Main Body, Door, Windows</td>
<td> 
Tree
Compulsory Components: Main body, Upper body</td>
</tr>
<tr>
<td> 
Street Light
Compulsory Components: Pillar, Lamp</td>
<td> 
Locomotive
Compulsory Components: Main Body, Chimney, Wheels, Windows</td>
</tr>
<tr>
<td> 
Goods Wagon
Compulsory Components: Main Body, Wheels</td>
<td> 
Passenger Wagon
Compulsory Components: Main Body, Wheels, Door, Windows</td>
</tr>
</table>

Note: The above images are rendered with a simple global light. Actual illumination implementation for this project is in the next phase, but if you want to have better looking objects, you may define a simple global light at this phase.

PHASE 3: The Crazy Machinist (Due Date: 04.01.2008)

In The Crazy Machinist Phase, you will implement some advanced graphics methods in order to enrich your environment with more realistic features.
The following specifications must be implemented in The Crazy Machinist Phase of your project:

- In this phase you will implement texture mapping to obtain a realistic looking world. You will be using texture mapping for two different goals; texturing the wagons and creating the skybox.
- The first goal is to texture a wagon (passenger wagon or goods wagon).
    - To use this functionality, the user must be at the walk-through mode in the mobile camera view and the animation must be stopped.
    - The user will select/pick a surface on the wagon in 3D. This surface may be member of one of the listed compulsory components of the wagon; main body, door, window.
    - The user will select a texture. For this purpose, you can provide a list of textures or you can provide a mechanism to enter a texture name that is available in a previously specified directory.
    - The selected texture will be mapped onto the selected surface.
- The second goal is to create a skybox. A skybox is a wide texture mapping, a cube or a sphere, that surrounds the entire world, and is mapped at infinitely far distances. Wherever the camera looks at, it sees the texture, which creates the illusion that there's the sky above. Note that the skybox will look realistic only if the texture is mapped to infinitely far distances, so that nothing can ever reach the sky. In the below figure you can see an example skybox and its mapping onto a cube.



- As stated in the general issues title, you have to preserve all the capabilities of the previous phases in this phase. For example, a change that has been done in 2D view of Phase 1 must still be reflected to other views at the same time.
- In this phase, the user must be able to change the state of the switches in 3D. This functionality will be accomplished at the walk-through mode in the mobile camera view.

- In many modern graphics applications, objects are represented by complex models which have been created in 3D modeling tools. In this phase, apart from the objects that you have used in the previous phases, you will also be adding some objects with complex models. In this project, we will name such objects as extra objects.
    - For this purpose 3ds format will be used. In order to load and render objects that are stored in this format, you will be provided a version of Object3DS library.
    - You will also be provided some sample .3ds object models. If you wish, you are free to use any other .3ds objects that you find on the internet at your own risk.
    - To use this functionality, the user must be at the walk-through mode in the mobile camera view and the animation must be stopped.
    - The user will choose an extra object and the object will be added to the environment. For this purpose, you can provide a list of extra objects or you can provide a mechanism to enter the model name of an extra object that is available in a previously specified directory.
    - The user can select a previously added extra object in 3D.
    - The user can translate, rotate or scale a selected extra object.
- Illumination can be considered to be one of the most effective components for realistic rendering in graphics applications. In this phase, you will illuminate a few light sources, which will give you a more realistic looking environment. There will be three types of light sources:
    - The Sun: A light source placed infinitely away from the world. The user must be able to turn on and off this source, which simulates night and day.
    - The Street Lights: They will be turned on automatically at night. The user must also have the option of turning all the street lights on or off at once.
    - The Train: It will have its own light placed at the front of the locomotive, moving with it. Once more, this light will be on by default at night, and the user must be able to turn it on or off.
- In order to achieve successful illumination, remember that you have to define normals for surfaces in your objects.

APPENDIX:

a) Libraries

During this project you may need the following resources about the libraries that you will use:

OpenGL:

The Red Book (The OpenGL Programming Guide)
main page: http://www.opengl.org/documentation/red_book/
html version: http://www.glprogramming.com/red/

The Blue Book (The OpenGL Reference Manual)
main page: http://www.opengl.org/documentation/blue_book/
html version: http://www.glprogramming.com/blue/

GLUT:

API can be found at:
http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

GLUI:

Main page of GLUI: (a manual including API is provided in this webpage)
http://glui.sourceforge.net/

Object3DS:

In fact, this is not complex library, but a wrapper class. It provides the ability to load and render "*.3ds" files. 3ds is a widely accepted format for using 3D models in computer graphics applications.

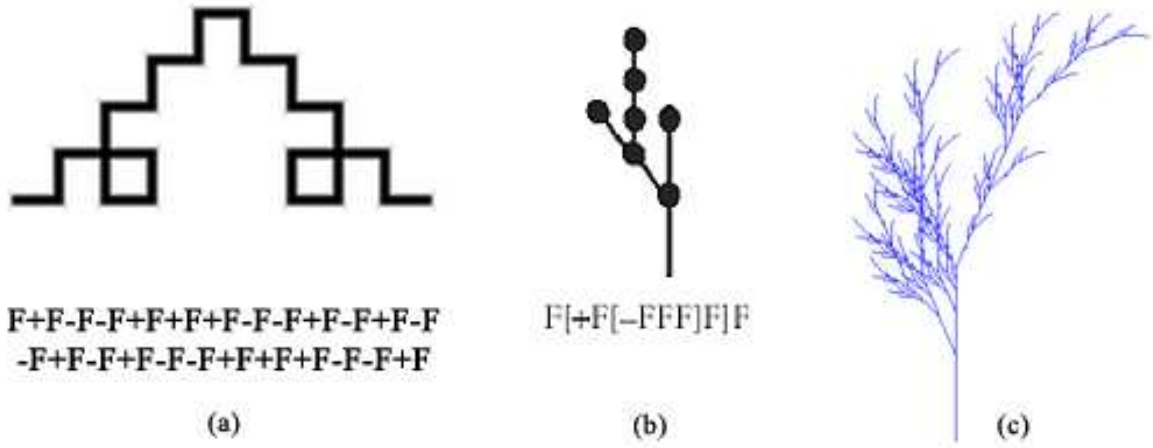
You can find more information about Object3DS in:
http://ironduke.cs.gsu.edu/gso_classes/OpenGL/aquilio/3ds.html
API:
http://ironduke.cs.gsu.edu/gso_classes/OpenGL/aquilio/reference.html

You will be provided an appropriate version of Object3DS by your assistants.

b) L-Systems and Turtle Graphics

"An L-system or Lindenmayer system is a formal grammar (a set of rules and symbols) most famously used to model the growth processes of plant development, but also able to model the morphology of a variety of organisms. L-systems can also be used to generate self-similar fractals such as iterated function systems" [1]

L-systems produce strings which then need to be geometrically/graphically interpreted in order to create graphical images. An example string is "F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F+F-F+F-F-F+F" and the image created from this string is given in the below figure on the left (a). [1]



For such interpretation, one of the most common methods is the turtle graphics. Turtle graphics translates each symbol on the input string (defined by L-system) to a drawing command for the display. Actually that is like a cursor; moving, turning and performing some drawing in your display (that's why they call it a turtle).

In this project we will be using our own variation of L-systems and turtle graphics. The related symbology is given in the below table:

Symbol	Turtle Graphic Interpretation in this project
t	Move forward a distance L while drawing a track
+	Turn left (counterclockwise) by angle 15^o
-	Turn right (clockwise) by angle -15^o
[	Create a switch and start a branch Push the current matrix to stack
]	End the branch Pop top of the stack and make it current matrix
e	End of L-system string

By using this symbology you will save and load the railroad definition for this project in a file named as railroad.txt. For this purpose, we will be using a common file format which is explained below:

- On a single line, two numbers separated by a blank representing x and y coordinates of the initial track of the railroad
- On a single line, a string representing the L-system definition for the saved railroad.
- On n lines, n user defined connection definitions; each represented by two integers of track IDs.

Track ID is the order of each track ('t' in our symbology) starting from 0 in the string representing the railroad. The following table illustrates the track IDs:

String	**t**	**t**	**+**	**t**	**t**	**t**	**-**	**t**	**t**	**+**	**t**	**+**	**t**	**[**	**t**	**]**	**+**	**t**	**e**
Track ID	0	1		2	3	4		5	6		7		8		9			10	

We will be providing sample railroad definition files for you and we can also try our own railroad definitions in your programs, during the demonstrations.

Syntactic Notes on symbology:

- There can be only one 'e' in the string.
- There can be only 1 consecutive element from the set {+, –}. (e.g. 't+-t' is not a valid string)
- There can be only 1 consecutive element from the set {[,]}. (e.g. t[[t]t]t is not valid)

References for appendix b:

[1] Wikipedia, L-system,
http://en.wikipedia.org/wiki/L-system
[2] Wikipedia, Turtle Graphics,
http://en.wikipedia.org/wiki/Turtle_graphics#Turtle_geometry_and_programming
[3] Turtle Graphics and L-Systems,
http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node18.html
[4] R. Parent, "Computer Animation: Algorithms and Techniques", 1st edition, Morgan Kaufmann, 2002, Pages 275-283